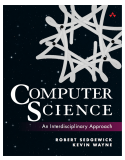
Hello World in Java (Linux)

This document instructs you on how to set up a Java programming environment for your Linux computer. It also provides a step-by-step guide for creating and compiling a Java program in *IntelliJ* and executing it from the command line.



You will need a 64-bit version of Linux.

Warning:

Beta version of instructions. Please send bug reports to wayne@princeton.edu.

Todo: replace Mac OS X screenshots with Linux screenshots.

1. Install Java

You will use *Java SE Development Kit 11* (*JDK 11*).

Note

Skip this step if you already have *JDK 11* installed.

- Log in to the user account in which you will be programming. Your account must have Administrator privileges and you must be connected to the Internet.
- Launch your shell. We'll assume that the command prompt looks like the following (though your command prompt will likely differ):

  ```
  ~>
  ```

  The symbol ~ is shorthand for your home directory.
- Install the *Java SE Development Kit 11*, either from OpenJDK or Oracle. Many Linux distributions (such as Ubuntu 18.04 or 20.04) include *OpenJDK 11* by default, so you can skip this step. Otherwise, use your Linux distribution's package manager (see the first FAQ) to install *OpenJDK 11*. For example, here are the commands for Ubuntu 16.04:

  ```
  ~> sudo add-apt-repository ppa:openjdk-r/ppa
  ~> sudo apt-get update
  ~> sudo apt-get install openjdk-11-jdk
  ```
- To confirm that Java 11 is installed, type the following commands:

  ```
  ~> javac -version
  javac 11.0.7

  ~> java -version
  openjdk version "11.0.7" 2020-04-14
  ```

```
OpenJDK Runtime Environment (build 11.0.7+10-Ubuntu)
OpenJDK 64-Bit Server VM (build 11.0.7+10-Ubuntu, mixed mode, sharing)
```

It's important that the Java version numbers match and that you see the number `11`, but the rest is not critical.

2. Install Command-Line Tools

Next, you will install our textbook libraries, *SpotBugs*, *PMD*, and *Checkstyle* to `/usr/local/lift` and associated wrapper scripts to `/usr/local/bin`.

- Type the following commands:

  ```
  ~> cd /usr/local
  /usr/local/> sudo curl -O "https://lift.cs.princeton.edu/java/linux/lift-cli.zip"
  /usr/local/> sudo unzip lift-cli.zip
  /usr/local/> sudo rm lift-cli.zip
  ```

  The command `curl` downloads files from the web.

- To confirm that the command-line tools are installed, type the following command:

  ```
  ~> java-introcs StdAudio
  ```

  You should hear an A-scale.

You should not need to update any shell configuration files (such as `.bashrc`) or set any environment variables (such as `JAVA_HOME` or `CLASSPATH`). For reference, here are our recommended shell configuation files for Bash:

- .bashrc
- .bash_profile
- .inputrc

3. Install IntelliJ

Now, you will install *IntelliJ*.

- Download and install IntelliJ IDEA, Community Edition 2020.1 for Linux. Use all of the default options.
- Download our *IntelliJ* preferences to your home directory.

  ```
  ~> cd
  ~> rm -rf .cache/JetBrains/IdeaIC2020.1
  ~> rm -rf .config/JetBrains/IdeaIC2020.1
  ~> rm -rf .local/share/JetBrains/IdeaIC2020.1
  ~> curl -O "https://lift.cs.princeton.edu/java/linux/IdeaIC2020.1.zip"
  ~> unzip IdeaIC2020.1.zip
  ~> rm IdeaIC2020.1.zip
  ```

  Warning

This will overwrite any previous *IntelliJ 2020.1* settings with our novice-friendly settings.

4. Open a Project in IntelliJ

You will develop your Java programs in an application called *IntelliJ IDEA, Community Edition*.

IntelliJ organizes Java programs into *projects*. In our context, each project corresponds to one programming assignment. A typical project contains Java programs, associated data files, and course-specific settings (such as compiler options, style rules, and textbook libraries).

- Download the project for your programming assignment to a convenient location (such as the Desktop).

  [sample project for COS 126 (Princeton)]

  - hello.zip

  [sample project for COS 226 (Princeton)]

  - percolation.zip

  [sample project for *Computer Science: Programming with a Purpose* (Coursera)]

  - hello.zip

  [sample project for *Algorithms, Part I* (Coursera)]

  - hello.zip
  - percolation.zip

  Unzip the zip file using the following command:

  ```
  ~>  unzip -d hello hello.zip
  ```

  This creates a project folder with the name of the corresponding programming assignment (such as `hello` or `percolation`). Delete the zip file.

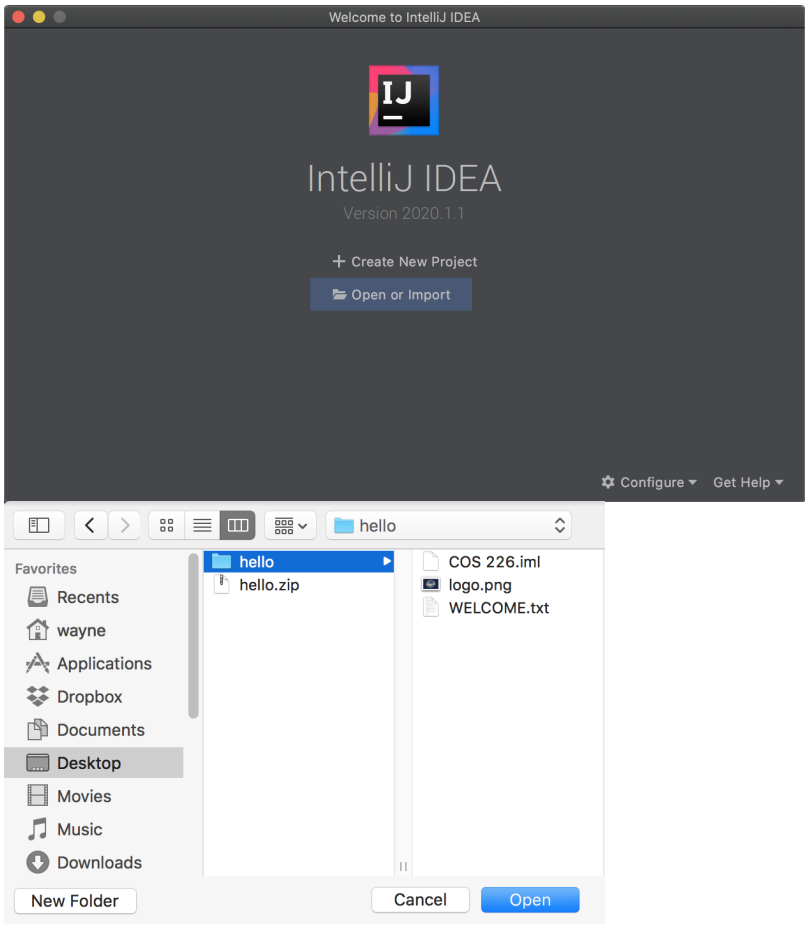
  Warning

  The project folders contain course-specific information. Be sure to download the one corresponding to your institution and course.

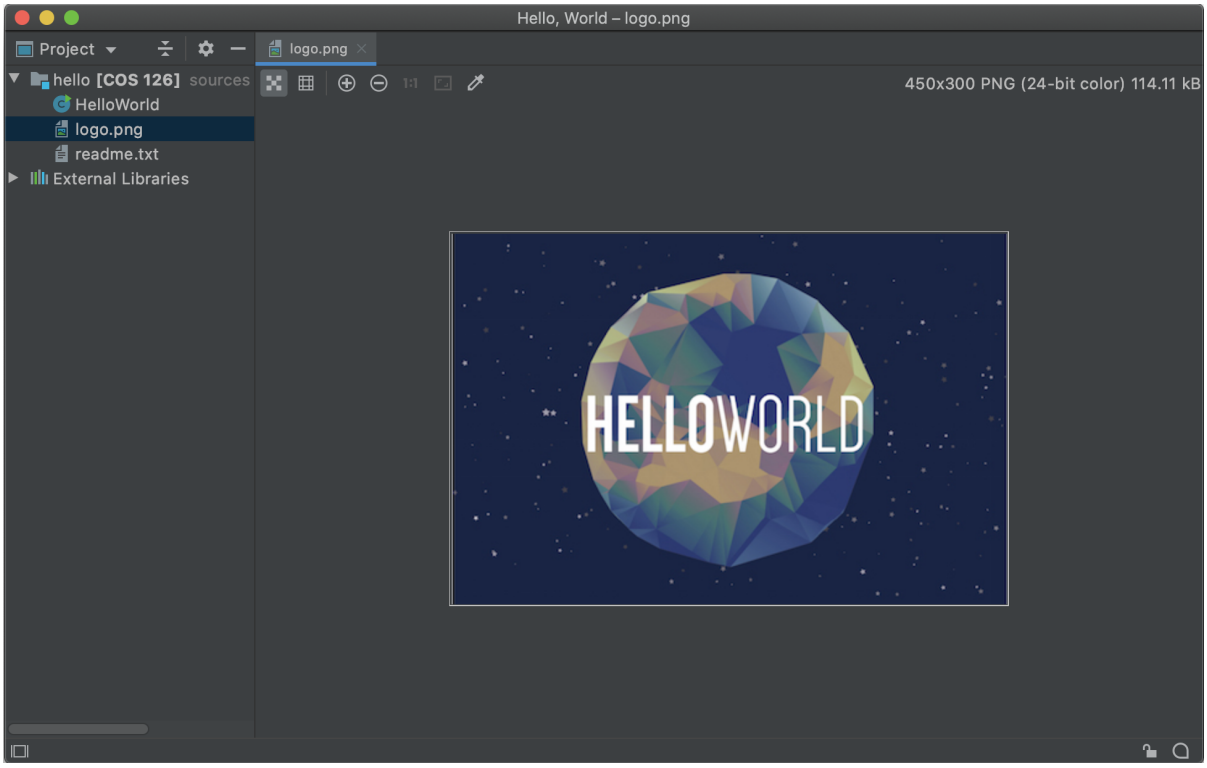
- Launch *IntelliJ*.
- When you launch *IntelliJ* for the first time,
  - *IntelliJ* may display the JetBrains privacy policy. Scroll down and *Accept*.

- *IntelliJ* may ask if you want to send anonymous usage statistics to JetBrains. Choose your preferred option.

- To open a project from the *Welcome screen*, click **Open** and select the project folder.



You should see an assignment logo (in the main editor window) and a list of project files (in the *Project View* sidebar at left).



When you launch *IntelliJ* for the first time, it may take a minute or two to index your files; some features (such as auto importing) will be unavailable until this process completes.

Warning

Do not select **Create New Project**; this option is intended for advanced programmers. Also, always use **Open** with a project folder, not an individual file.

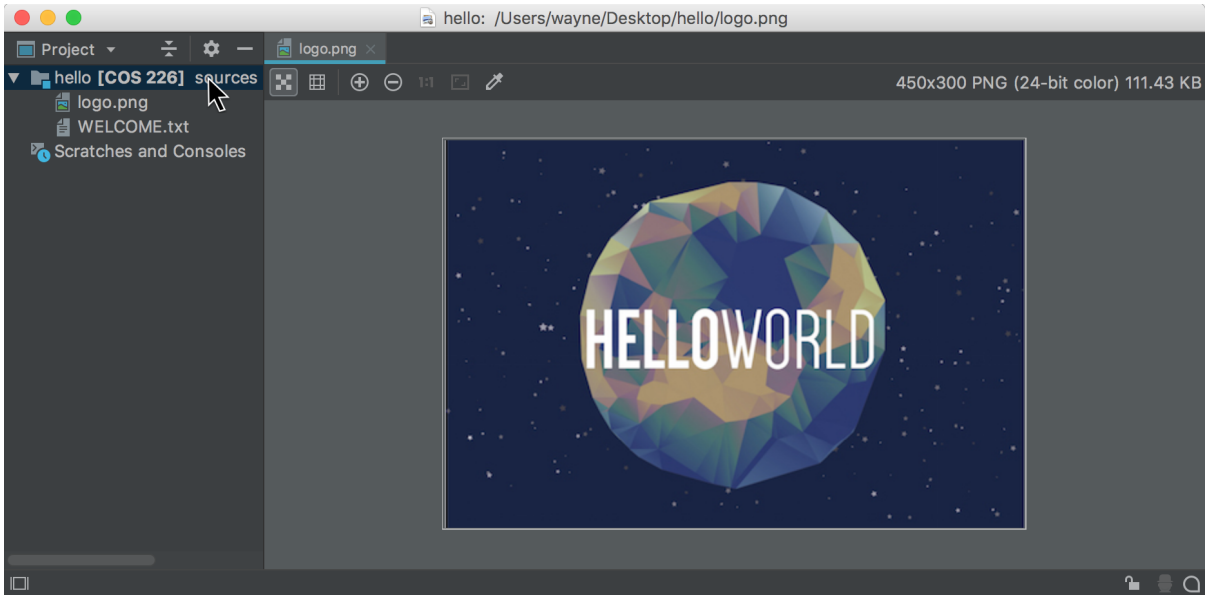
- You will need to manually configure the Platform SDK. To do so,
  - Navigate to *File → Project Structure → Platform Settings → SDKs*.
  - Click the + symbol (top left) to add an SDK.
  - Locate an SDK. A typical location for a Java SDK on Linux is `/usr/lib/jvm/java-11-openjdk-amd64/`.
  - Use the shorthand name suggested by *IntelliJ* (e.g., 11 for version 11.0.7).
- When you are finished working, select the menu option **File → Exit** to exit *IntelliJ*. The next time you launch *IntelliJ*, your recent projects will appear in the *Welcome screen* for easy access.

5. Create a Program in IntelliJ

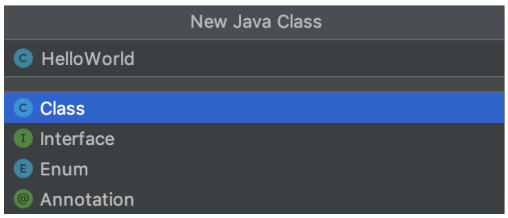
Now you are ready to write your first Java program. *IntelliJ* features many specialized programming tools

including line numbering, syntax highlighting, bracket matching, auto indenting, auto formatting, auto importing, variable renaming, and continuous code inspection.

- To create a new Java program:
  - Re-open *IntelliJ* and the project (if you closed it in the previous step).
  - Click the project name in the *Project View* sidebar (at left), so that it becomes highlighted.



  - Select the menu option **LIFT → New Java Class**. When prompted, type **HelloWorld** for the *Name* and click **OK**.

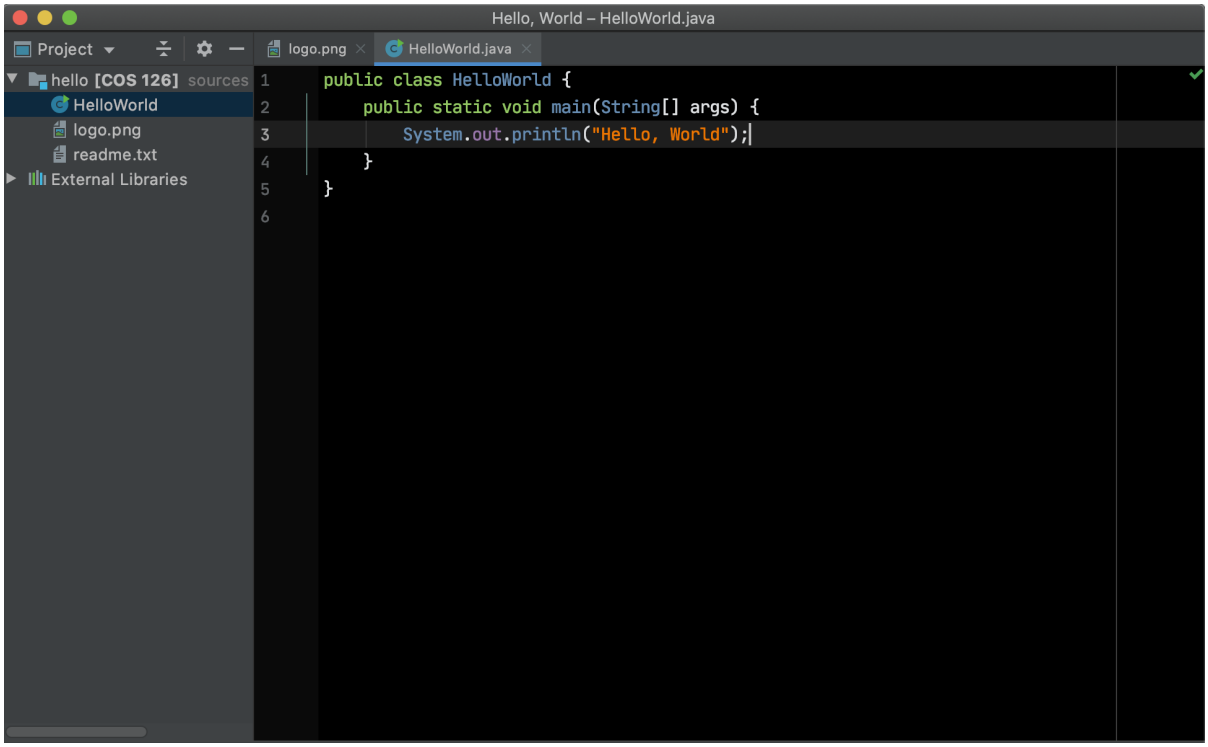


- In the main editor window, complete the Java program `HelloWorld.java` exactly as it appears below. (*IntelliJ* generates the gray boilerplate code automatically, along with the course header block comment.)

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

  If you omit even a semicolon, the program won't work.

- As you type, *IntelliJ* highlights different syntactic elements in different colors. When you type a left bracket, *IntelliJ* adds the matching right bracket. When you begin a new line, *IntelliJ* indents it.



- To save the file, select the menu option **File → Save All (Ctrl + S)**. When you save the file, *IntelliJ* re-formats it (if necessary).

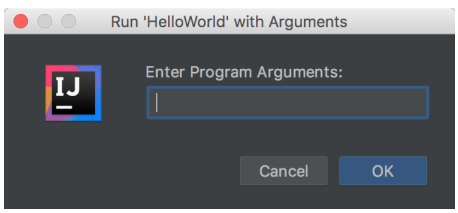
6. Compile and Execute the Program (from IntelliJ)

Now, it is time to *execute* (or *run*) your program. This is the exciting part, where your computer follows the instructions specified by your program. Before doing so, you must *compile* your program into a form more amenable for execution on a computer.

- Select the program that you wish to compile and execute in the the *Project View* sidebar. The program should now appear in the main editor window.
- To *compile* your program, select the menu option **LIFT → Recompile 'HelloWorld.java' (Ctrl + B)**. If the compilation succeeds, you will receive confirmation in the status bar (at bottom).

If the compilation fails, a *Recompile panel* will open up (at bottom), highlighting the compile-time errors or warnings. Check your program carefully for typos, using the error messages as a guide.

- To *execute* your program, select the menu option **LIFT → Run 'HelloWorld' with Arguments (Ctrl + E)**. Since this program takes no command-line arguments, click **OK**.



You should see the output of the program (in white), along with a message that the program finished normally (with exit code 0).

Tip

Use the **LIFT** menu to compile and execute your program from *IntelliJ*. The **Build** and **Run**menus support additional options for advanced programmers.

Also be sure that the main editor window is active before using the **LIFT** menu (e.g., by clicking the code you want to compile or execute).

7. Compile and Execute the Program (from the command line)

The *command line* is a simple and powerful mechanism for controlling your programs (e.g., command-line arguments, file redirection, and piping). *IntelliJ* supplies an *embedded terminal*for easy access to the command line.

- Select the menu option **View → Tool Windows → Terminal (Alt + 2)**.
- This will launch a *Bash terminal* where you type commands. You will see a *command prompt* that looks something like this:

  ```
  ~/hello>
  ```

  The `~/hello` is the current working directory, where ~ is shorthand for your home directory.

- To *compile* your program, type the following `javac` command. More specifically, type the text in yellow that appears on the same line as the command prompt.

  ```
  ~/hello> javac HelloWorld.java
  ~/hello>
  ```

  Assuming that the file `HelloWorld.java` is in the current working directory, you should not see any compile-time errors or warnings.

- To *execute* your program, type the following `java` command:

  ```
  ~/hello> java HelloWorld
  Hello, World
  ```

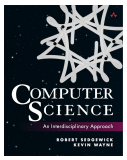
  You should see the output of your program beneath the line on which you typed the command.

Tip

Typically, you should **compile from *IntelliJ*** (because *IntelliJ* highlights the lines on which any compile-time errors or warnings occur) and **execute from the command line**(because the command line makes it is easy to specify command-line arguments and use file redirection).

8. Textbook Libraries (from the command line)

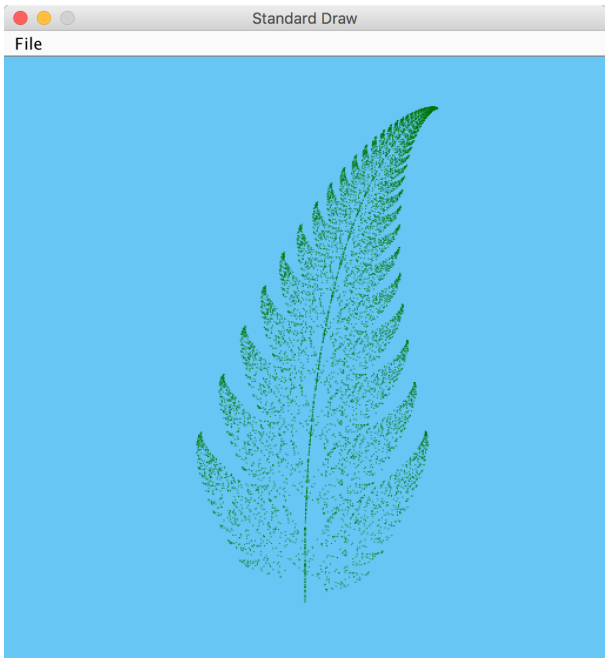
To make our textbook libraries accessible to Java from the command line, you will use our wrapper scripts.



- **Computer Science: An Interdisciplinary Approach (including COS 126 students).** The program Barnsley.java uses our *standard drawing* and *standard random* libraries in `stdlib.jar` to draw a Barnsley fern. First download Barnsley.java. Then, use your system's file manager (such as *Nautilus*) to move it to a project folder (such as `hello`). Finally, to *compile* and *execute*it, type the following commands in the terminal:

  ```
  ~/hello> ls
  Barnsley.java  COS 126.iml    WELCOME.txt    logo.png
  ~/hello> javac-introcs Barnsley.java
  ~/hello> java-introcs Barnsley 10000
  ```

  When you execute the program, a *standard drawing* window will appear and an image like this one will be generated, one point at a time:

  To get your command prompt back, close the *standard drawing* window.

- **Algorithms, 4th Edition (including COS 226 and Coursera students).**The program CollidingDisks.java uses various libraries in `algs4.jar` to simulate the motion of *n* disks subject to the laws of elastic collision. First download CollidingDisks.java Then, use your system's file manager (such as *Nautilus*) to move it to a project folder (such as `percolation`). Finally, to *compile* and *execute* it, type the following commands in the terminal:

```
~/hello> ls
CollidingDisks.java  COS 226.iml    WELCOME.txt    logo.png
~/hello> javac-algs4 CollidingDisks.java
~/hello> java-algs4 CollidingDisks 20
```

When you execute the program, a *standard drawing* window will appear with an animation of 20 colliding disks. To get your command prompt back, close the *standard drawing* window.

Frequently Asked Questions

Expand All Collapse All

Licensing FAQ

How is the software licensed?

Linux FAQ

My distribution of Linux is { Gentoo, Debian, Ubuntu, Fedora, Red Hat, SuSE, Mandriva, or Slackware }. How should I modify the instructions?

Java FAQ

Can I use a vendor and version of Java other than OpenJDK 11?

How can I check which version of Java is installed (and where it is installed)?

IntelliJ FAQ

How does this custom version of *IntelliJ* different from the standard one?

What are the most important *IntelliJ* menu options to remember?

Any special characters to avoid when naming IntelliJ projects or files?

How can I create a new project in IntelliJ?

Can I use a version of IntelliJ that is more recent than 2020.1.1?

How I can I restore the original IntelliJ settings (instead of the abbreviated novice-friendly ones)?

Command-Line / Embedded Terminal FAQ

When I compile or execute a program from the command line that uses one of the textbook libraries, I get an error that it cannot find the library. How can I fix this?

How should I configure Bash?

How do I break out of a program in an infinite loop?

How do I specify EOF to signal that standard input is empty?

How can I run SpotBugs, PMD, and Checkstyle from the command line?

How do I verify that `/usr/local/bin` is in my `PATH` environment variable?